10. ročník Žilina 2. – 4. 7. 2018
Otvorený softvér vo vzdelávaní, výskume a v IT riešeniach
osscconf.soit.sk
OSS Conf
OSSConf 2025: 1–9

Generování pseudonáhodného čísla po cifrách

PAVEL STŘÍŽ (CZ)

Abstrakt. Článek představuje jednoduchý algoritmus na generování pseudonáhodného čísla po cifrách (brány zleva doprava). Interval bere v rozsahu $[0;N-1]$, kde N se bere jako jistý počet možností, nejčastěji kombinací, které jsou indexovány od jedničky. Autor zmiňuje svůj postřeh, že je potřeba jen cca prvních deset cifer, zbytek generovaných cifer lze rozhodit na další výpočetní stroje, protože se volí čísla v rozsahu [0;9] bez dodatečné podmínky. Princip algoritmu by byl stejný i pro binární čísla. Zdrojové kódy jsou v článku přidány, pro Python3 a pro R. V závěru příspěvku autor zmiňuje nastavení svého oblíbeného editoru SciTE, tedy aby zdrojový kód v R byl barevně zvýrazněn a bylo možné jej z editoru spouštět.

Klíčová slova. PRNG, R, Python, C, SciTE.

PSEUDORANDOM NUMBER GENERATION BY DIGITS

Abstract. The article introduces an simple algorithm which generates a pseudorandom number by digits (from left to right). The interval of the number is $[0;N-1]$, where N is a number of certain situations, most often combinations (indexed from 1). The author states that we only need about first 10 digits, then we can distribute problem to different computers because we generate digits from interval [0;9] without any condition. The idea of the algorithm would be the same for binary numbers. The source code is included, in Python3 and in R. In the conclusion for user's convenience, the author mentiones his gained experience in setting up the SciTE editor to activate syntax highlighting and turn on the option of running the R code within the editor.

Keywords. PRNG, R, Python, C, SciTE.

1. O problému

Při řešení Rubikovy kostky: výpočtu kombinací, výběru jedné z nich a zobrazení Rubikovy kostky, a obráceně, po zobrazení jisté Rubikovy kostky zjistit pořadové číslo kombinace, jsem zjistil, že počet kombinací u větších Rubikových kostek dosahuje obřích čísel. U řešené virtuální kostky (světový rekord, hrana délky $n = 2^{16}$) se dostáváme na počet kombinací nad 16 miliard cifer. To už není na počítání, řekl bych.

U Rubikovy kostky už jen pro délku hrany $n = 2^{11} = 2048$ dostáváme počet možností N o 16257517 cifrách, můžete si se svém oblíbeném programu na

výpočty ověřit. V závěru článku na str. 8 představuji řešení pro Python.

$$N = 7! \cdot 3^6 \cdot (24 \cdot 2^{10} \cdot 12!)^{n \bmod 2} \cdot 24!^{\left\lfloor \frac{n-2}{2} \right\rfloor} \cdot \left(\frac{24!}{4!^6}\right)^{\left\lfloor \left(\frac{n-2}{2}\right)^2 \right\rfloor}$$

Dokonce i zamíchat takovou kostku se touto cestou už skoro nedá, musí se na to jít bez výpočtu možností, či zkusit rozhodit skupiny barev, jak toho využívá program RCube. Zamíchání Rubikovy kostky dle sekvence [FBLRUD] není totiž rovnoměrně náhodné.

I tak jsem si říkal, že by bylo zajímavé si vygenerovat pseudonáhodné číslo v intervalu $[0; N-1]$, kde N je počet možností (indexován od jedničky).

2. První úvaha

Asi nejjednodušší cesta je vzít počet cifer takového obrovského čísla. Vygenerovat si sérii cifer 0–9 a na konci srovnat s $N-1$. Je-li vygenerovaná hodnota menší či rovna $N-1$, uznáme ji, v opačném případě generujeme celou sekvenci znovu.

3. Druhá, ošklivá úvaha

Asi nejrychlejší výpočetní podvod je vzít první cifru a ponížit ji o jedničku. Na první cifře generovat v rozsahu [0;dopočtené], všechny ostatní cifry pak už v příjemném rozsahu [0;9]. Například u čísla 6855 dostáváme 0–5 na první cifře a 0–9 na zbylých, tedy dostáváme čísla v rozsahu 0000–5999. Bohužel za tu cenu, že nikdy nezískáme hodnotu v rozsahu 6000–6855. Úvahu, s úsměvem, opusťme jako nepatřičnou.

Pokud bychom měli hovořit o ošklivosti, tak ještě horší, byť na první pohled funkční případ, je užít na každé cifře interval nula až daná cifra, např. u hledání čísla [0;634] volit 0–6 na první cifře, 0–3 na druhé a 0–4 na třetí. Zdánlivě je vše v pořádku, ale to je omyl, např. číslo 552 nelze získat díky druhé cifře.

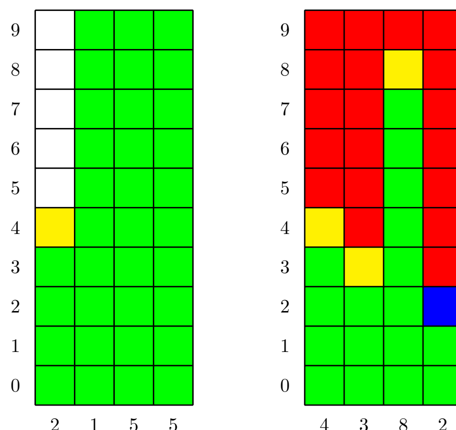
4. Lepší nápad

První úvaha je použitelná, ale časově náročná, neb řadu hodnot musíme vyřadit. Jistá myšlenka byla, jak by se tato situace dala zefektivnit. Algoritmus by se dal popsat slovy takto. Vnější cyklus jde přes cifry, zleva doprava (nejvyšší řád mocniny po nejnižší). Vygenerujeme číslo v intervalu [0;9] a srovnáme s cifrou na zkoumané pozici. Pokud je menší, u všech dalších cifer může být [0;9] a zmíněná podmínka není dále nutná. Pokud je cifra stejná, hodnotu přijmeme, ale podmínka zůstává zachována pro další cifry. A pokud je cifra vyšší, celý pokus rušíme a začneme znovu.

Výjimkou je 1. cifra, tam s jistotou víme, že generovaná cifra musí být v intervalu jedna až zkoumaná první cifra, nemusíme se tedy držet intervalu [0;9].

Poznámka. Kdybychom místo [0;9] užili interval nula až zkoumaná cifra při neaktivní podmínce, dostali bychom se do podmíněné pravděpodobnosti, to je nad rámec tohoto příspěvku. I tak je to slepá ulička.

Zde je ukázka hodnoty z [0;4282]. U čísla 2155, jsme u první cifry zjistili, že je menší než 4, ostatní mohou získat libovolné hodnoty. Naopak u čísla 4282 samotného jsme s podmínkou stále aktivní i po čtyřech cifrách. Například číslo 4400 bychom vyřadili na druhé cifře, 6000 a vyšší již na první cifře apod.



Například u čtyřciferného čísla (1000–9999) je nejhorší možnost právě 1000. Vyřadíme hodnoty 1001–9999 (cca 90 % hodnot), u čísla 9999 bychom nevyřadili pokus žádný. V závislosti na zvoleném N však můžeme říci, že v průměru vyřazujeme 45 % hodnot. Poněvadž ale vyřazení pokusu uděláme obvykle dřív než u posledních cifer, stálo za hřích zkusit simulaci, na které cifře ke zrušení podmínky sledování nižší hodnoty dochází. Tím se totiž zajistí, že už k dalšímu vyřazení nemůže dojít.

Cifra	Přij. abs.	Přij. rel.	Přij. abs. kum.	Přij. rel. kum.
1	8800861	0,8800861	8800861	0,8800861
2	1071983	0,1071983	9872844	0,9872844
3	114334	0,0114334	9987178	0,9987178
4	11499	0,0011499	9998677	0,9998677
5	1173	0,0001173	9999850	0,9999850
6	132	0,0000132	9999982	0,9999982
7	14	0,0000014	9999996	0,9999996
8	4	0,0000004	10000000	1,0000000
9	0	0,0000000	10000000	1,0000000
10	0	0,0000000	10000000	1,0000000
$\sum$	10000000	1,0000000	—	—

Ze simulovaných 10 miliónů čísel o deseti cifrách (hledané číslo jen jednou) vidíme, že jsme potřebovali prvních osm cifer. Další cifry byly o výběru z 0–9,

což se dá distribuovat na jiné výpočetní stroje. Při 100 miliónech čísel jsem se dostal na dva případy na deváté cifře.

Poněvadž se generování opakuje od začátku, když je generovaná hodnota při aktivní podmínce na cifře vyšší než hledaná, pro badatele by mohla být zajímavá tato tabulka, která zmiňuje počet pokusů než se generování celého čísla podařilo.

Pokusů	Případů	Pokusů	Případů
1	8927265	11	57
2	869600	12	23
3	148173	13	17
4	36748	14	7
5	11385	15	3
6	4017	16	1
7	1604	17	0
8	706	18	0
9	283	19	0
10	111	20	0

Nejlepší statistika pokusů je u čísla se samými devítkami (vždy první úroveň, není kde číslo vyřadit), nejhorší pak s číslem začínajícím jedničkou a zbytek nuly (dochází i na úroveň mnoha desítek pokusů).

Možností tohoto algoritmu je, že nám stačí generování bitů. Ze sekvence bitů si už libovolně velké číslo $N-1$ složíme. Princip metody je však stejný jako u čísel desítkové soustavy.

Pokud tedy generujeme číslo o tisících cifrách a mnohem větší, je nám už jedno, jestli na úvod uřízneme prvních osm, deset či dvacet cifer. Důležité pro nás bylo vědět, kde alespoň přibližně se máme pohybovat. S každou dodatečnou cifrou se řádově posouváme o řád u zrušení podmínky. 10 miliónů čísel u této simulace je osmiciferné číslo, tedy řádově počítáme hranici osm uřízlých cifer plus nějaká rezerva. Je to dané tím, že posun mezi ciframi bez zrušení podmínky zajišťuje jedna hodnota z deseti možných, tedy 10 %.

5. Poznámky k programům

Jednoduchý program byl prvně naprogramován v Pythonu3 a pak za pomoci `https://www.javainuse.com/py2r` převeden do R a za pomoci webové stránky `https://extendsclass.com/python-to-javascript.html` převeden do JavaScriptu. Zdrojový kód pro JavaScript zde neuvádím.

```
import random
hodnota=4383 # Indexování od 1 -- N.
cislo=str(hodnota-1) # Indexování od 0 -- N-1.
kolik=40 # Kolik hodnot si přeji vygenerovat.
print("Generuji",kolik,"hodnoty od nuly po",cislo,"...")
delka=len(cislo)
def generuj():
```

```
    while True:
        odCisla=0; retezec=""; doCisla=int(cislo[0])
        mensi=False; poradi=0; znovu=False
        while poradi<delka:
            nahodne=random.randint(odCisla,doCisla)
            doCisla=9
            if nahodne<int(cislo[poradi]): mensi=True
            if nahodne<=int(cislo[poradi]) or mensi:
                retezec+=str(nahodne)
            else:
                znovu=True; break
            poradi+=1
        if not znovu:
            nalezene=int(retezec)
            if nalezene<hodnota:
                print(retezec); return()
for _ in range(kolik): generuj()
```

Kdybychom v Pythonu3 chtěli skutečně pracovat s obřími čísly, nikoliv textovým řetězcem hned od začátku, užijeme:

```
import sys
sys.set_int_max_str_digits(0)
```

Při online konverzi jsem zjistil, že Python3 bere `return` i `return()`, výstup se mi u R zacyklil. Konvertítko chybně bralo „ě“, u „í“ zahlásilo chybu.

Pro badatele ve výpočetním prostředí R přikládám zdrojový kód.

```
# Generování PRN po cifrách...
hodnota <- 4383 # Indexování od jedničky, 1 -- N.
cislo <- as.character(hodnota - 1) # Indexování od nuly, 0 -- N-1.
kolik <- 40 # Počet chtěných cifer.
print(paste("Generuji", kolik,"hodnot(y) od nuly po", cislo, "..."))
delka <- nchar(cislo)
generuj <- function(){
  while(TRUE){
    odCisla <- 0; retezec <- ""
    doCisla <- as.numeric(substr(cislo, 1, 1))
    mensi <- FALSE
    poradi <- 0; znovu <- FALSE
    while(poradi < delka){
      nahodne <- sample(odCisla:doCisla, 1)
      doCisla <- 9
      if(nahodne < as.numeric(substr(cislo,poradi+1,poradi+1))){mensi <- TRUE}
      if(nahodne <= as.numeric(substr(cislo,poradi+1,poradi+1)) || mensi){
        retezec <- paste(retezec, nahodne, sep = "")
      } else { znovu <- TRUE; break }
      poradi <- poradi + 1
    }
    if(!znovu){
      nalezene <- as.numeric(retezec)
      if(nalezene < hodnota){
        print(retezec); return()
}}}}
for(invalue in 1:kolik){generuj()}
```

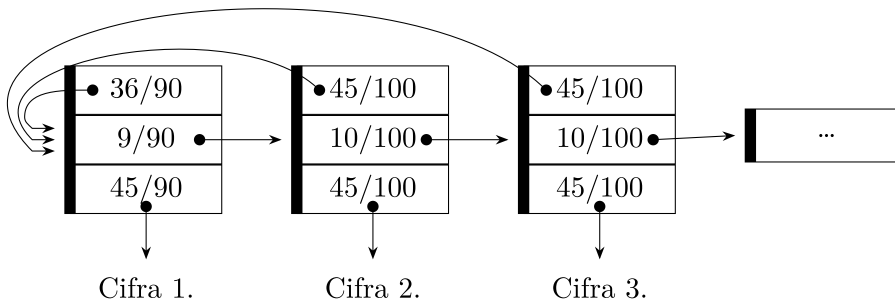
6. Simulujme!

Věřím tomu, že se jedná o zajímavou pravděpodobnostní úlohu výpočtu přesné hodnoty, s jakou pravděpodobností na cifře dojde k vyřazení podmínky; úloha však byla a je nad autorovy síly. Kdyby na to náhodou někdy, někdo došel, pochlubte se.

Zkusím však celou situaci zjednodušit. Na každé úrovni se rozhodujeme mezi třemi stavy. Pokus opakovat (hodnota vyšší než chtěná), posunout se na další úroveň (hodnota je stejná jako žádaná) a nakonec uzavřít experiment (zapadnutí čísla do úrovně, zbytek je už jen generování cifer 0–9, tedy pro nás nezajímavé). V tabulce je souhrn všech možností, šipka dolů je uložení úrovně, šipka vlevo opakovat pokus a šipka vpravo zvednutí úrovně.

Žádaná \ Generovaná	0	1	2	3	4	5	6	7	8	9
0	→	←	←	←	←	←	←	←	←	←
1	↓	→	←	←	←	←	←	←	←	←
2	↓	↓	→	←	←	←	←	←	←	←
3	↓	↓	↓	→	←	←	←	←	←	←
4	↓	↓	↓	↓	→	←	←	←	←	←
5	↓	↓	↓	↓	↓	→	←	←	←	←
6	↓	↓	↓	↓	↓	↓	→	←	←	←
7	↓	↓	↓	↓	↓	↓	↓	→	←	←
8	↓	↓	↓	↓	↓	↓	↓	↓	→	←
9	↓	↓	↓	↓	↓	↓	↓	↓	↓	→

Tím dostáváme 45 situací (opakovat), 10 situací (posunutí o úroveň) a dalších 45 situací (uzavření čísla na dané úrovni), zkráceně 45-10-45. Jedinou výjimkou je první úroveň, kdy pro n-ciferné číslo předpokládáme, že nezačíná nulou, tím odpadá první řádek a dostáváme 36-9-45.



Pro Céčkaře něco na závěr. Tímto pokusem jsem se dostal na jeden případ na desáté cifře, a asi nejpřesněji, jak jsem to jen dovedl.

```
// gcc -o 45-10-45 45-10-45.c && ./45-10-45
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
long maxH=1000000000;
char uroven, nahodna;
long urovne[16]; for (char k=0; k<16; k++) {urovne[k]=0;}
srandom(0);
for (long hodnot=0; hodnot<maxH; hodnot++) {
    uroven=0;
    while (1) {
        uroven++;
        if (uroven>1) {
            nahodna=random()%100;
            if (nahodna<45) {urovne[uroven]++; break;}
            if (nahodna>54) {uroven=0;}
        } else {
            nahodna=random()%90; // zásah
            if (nahodna<45) {urovne[uroven]++; break;}
            if (nahodna>53) {uroven=0;} // zásah
        } // if
    } // while
} // for
for (char k=1; k<16; k++) {printf("%d %ld %0.016f\n",
    k, urovne[k], (double)urovne[k]/maxH);}
return 0;
}
```

Tiše sedím, sázím tento článek, a říkám si, že bych nemusel čísla takto posílat po jednom, že je to neefektivní, ale rovnou zaslat celý blok. A poposouvat čísla do hloubky také v blocích, kam jen to jde, vždyť struktura je známá. A z hodnot zasílaných zpět začít znovu, dokud se nám nepodaří vše rozhodit. Zde je na závěr kód pro Pythonisty.

```
import math
zaklad=10000000000000000000; varka=zaklad
urovne=[0]*21; zbytky=[0]*21
def hloubeji(kolik,uroven): # Rekurze
    global varka
    kolik=kolik+zbytky[uroven]
    zpet=math.floor(kolik*45/100)
    poslidal=math.floor(kolik*10/100)
    posliuroven=math.floor(kolik*45/100)
    urovne[uroven]+=posliuroven
    zbytky[uroven]=kolik-zpet-poslidal-posliuroven
    varka=varka+zpet
    if poslidal>0: hloubeji(poslidal,uroven+1)
while True:
    uloz=math.floor(varka*45/90+varka*36/90*45/(9+45))
    dal=math.floor(varka*9/90+varka*36/90*9/(9+45))
    varka=varka-dal-uloz # nová várka
    urovne[1]+=uloz
    if dal>0: # Popošli blok čísel...
        hloubeji(dal,2)
    else:
        # Uzavření výpočtů...
        zbytky[1]=varka; varka=0; break
```

```
soucet=0
for uroven in range(1,len(urovne)):
    urovne[uroven]+=zbytky[uroven] # Závěrečné přičtení zbytků k úrovni.
    soucet+=urovne[uroven]
    print(uroven, urovne[uroven],
        format(urovne[uroven]/zaklad,"0.020f"),
        soucet, format(soucet/zaklad,"0.020f"))
```

Pole `zbytky` je pomůcka, jak si dávat pozor na zaokrouhlování čísel. Na úplný závěr se přičtou k dané úrovni, ale jedná se už jen o jednotky. Myslím, že struktura rozhození do úrovní přes všechny případy je už čitelná. Matematik by řekl, že se jedná o (ne)konečné řetězové zlomky.

Cifra	Přijatých absolutně	Přijatých relativně	Přij. abs. kumulovaně	Přij. rel. kumulovaně
1	9090909090909091044	0,9090909090909090606	9090909090909091044	0,9090909090909090
2	818181818181818044	0,0818181818181818038	9909090909090909088	0,9909090909090909
3	81818181818181812	0,0081818181818181807	9990909090909090900	0,9990909090909091
4	8181818181818182	0,0008181818181818181	9999090909090909082	0,9999090909090908
5	818181818181819	0,0000818181818181819	9999909090909090901	0,9999909090909090
6	81818181818181	0,0000081818181818181	9999990909090909082	0,9999990909090908
7	8181818181819	0,0000008181818181819	9999999090909090901	0,9999999090909090
8	818181818183	0,0000000818181818183	9999999909090909084	0,9999999909090908
9	81818181819	0,0000000081818181819	9999999990909090903	0,9999999990909090
10	8181818181	0,0000000008181818181	9999999999090909084	0,9999999999090909
11	818181820	0,0000000000818181820	9999999999909090904	0,9999999999909090
12	81818182	0,0000000000081818182	9999999999990909086	0,9999999999990909
13	8181819	0,0000000000008181819	9999999999999090905	0,9999999999999090
14	818182	0,0000000000000818182	9999999999999909087	0,9999999999999908
15	81819	0,0000000000000081819	9999999999999990906	0,9999999999999991
16	8182	0,0000000000000008182	9999999999999999088	0,9999999999999998
17	819	0,0000000000000000819	9999999999999999907	1,0000000000000000
18	83	0,0000000000000000083	9999999999999999990	1,0000000000000000
19	9	0,0000000000000000009	9999999999999999999	1,0000000000000000
20	1	0,0000000000000000001	10000000000000000000	1,0000000000000000
$\sum$	10000000000000000000	1,0000000000000000000	—	—

7. Implementace

Kdybychom něco takového skutečně potřebovali a i při všech těchto znalostech, nejlepší je se riziku vyhnout. Riziko v tomto případě znamená, že se některý blok čísel bude brát v ohledu vícekrát. Buď tedy ať výpočetní stroj generuje všechny cifry, nebo po ukončení podmínky, ať si jiným strojům řekne *kolik* cifer ještě chybí. Než to s rizikem dělat tak, že si na začátku řekneme, že budeme generovat např. 10, 20 či 30 cifer na jednom stroji a zbytek na jiných.

Zde je výpočet počtu cifer ze začátku článku ze str. 2. Funguje to v Pythonu verze 3.10.12. Převod na celá čísla, `int()`, v posledním činiteli je tam záměrně, abychom po operaci dělení nepracovali s desetinným číslem, to má svá omezení.

```
import math
import sys; sys.set_int_max_str_digits(0)
n=2**11 # Základní Rubikova kostka by byla n=3.
x=math.factorial(7) * 3**6 * \
  (24 * 2**10 * math.factorial(12))**(n%2) * \
```

```
  math.factorial(24)**(math.floor((n-2)/2)) * \
  (int(math.factorial(24)/(math.factorial(4)**6)))** \
  (math.floor(((n-2)/2)**2))
print("Délka hrany =", n, ", počet cifer =", len(str(x)))
```

Poznámka. Tato práce s velkými čísly (angl. Arbitrary Number Precision) je stále otevřený problém v C a C++. Uživatel musí sáhnout do jiných zdrojů mimo oficiální knihovny či si to musí naprogramovat sám. V ukázce v Pythonu či v Javě (knihovna `BigNumber`) to už takový problém není.

8. Poznámky k editoru SciTE místo Závěru

Je to k nevíře, ale do řešení této úlohy jsem svůj oblíbený editor (SciTE) na editaci a spuštění R kódu nepoužil. Instalujeme si jej případně přes (Ubuntu):

```
sudo apt install scite
```

spuštění přes `scite`, nebo přes:

```
flatpak install flathub org.scintilla.SciTE
```

spuštění přes `flatpak run org.scintilla.SciTE`.

Abychom dostali formátovaný soubor napsaný v R, musíme na konci souboru `/usr/share/scite/SciTEGlobal.properties` vyhodit R z filtru (zapsán malým písmenem er). Předdefinovaný v menu v nabídce programů není.

Pro spuštění (klávesa F5) jsem si v `~/.SciTEUser.properties` přidal:

```
command.go.$(file.patterns.r)=Rscript $(FileNameExt)
```

Nemohu to na 100 % potvrdit ani vyvrátit, ale v nové verzi SciTE lze přes Ctrl a plus či mínus zvětšovat a zmenšovat písmo, to si v dřívějších verzích nevybavuji. Po bližším zkoumání zjišťuji, že to tím není. Funguje to s plus a mínus na numerické klávesnici, nikoliv s těmi ze základní sady písmen. To jsem si vylepšil v `~/.SciTEUser.properties` přidáním (2333 je kód pro ZoomIn, 2334 pro ZoomOut, to jsem vyčetl ze souboru `CommandValues.html` z dokumentace tohoto programu):

```
user.shortcuts=Ctrl++|2333|Ctrl+-|2334|
```

Vítejte v eRkovém a výpočetním světě!

Kontaktní adresa

Ing. Pavel Stříž, Ph.D., U Škol 940, Bučovice, okres Vyškov, 685 01, Česká republika,
E-mailová adresa: pavel@striz.cz